Learning Ionic

With this book, you will learn hybrid mobile application development using Ionic. This book uses Cordova 5.0.0, Ionic CLI 1.5.0, and Ionic 1.0.0 to explain the concepts and build apps.

To begin with, the book helps you understand where Ionic fits in today's world. Then, you will dive deep into Ionic CSS components and Ionic-AngularJS directives and services. You will also examine themeing Ionic apps using the built-in SCSS setup.

Next, you will learn to build an Ionic client for a secure REST API, where you will implement user authentication and token-based development. Cordova and ngCordova will be explored, and you will learn how you can integrate device-specific features such as a camera and Geolocation with an Ionic app. We will wrap up the book by building a messaging app, which deals with talking to Firebase (a real-time data store), as well as device features.

By the end of this book, you will be able to develop a hybrid mobile application from start to finish, and generate device-specific installers.

Who this book is written for

This book is intended for those who want to learn how to build hybrid mobile applications using Ionic. It is also ideal for people who want to explore themeing for Ionic apps. Prior knowledge of AngularJS is essential to complete this book successfully.

What you will learn from this book

- Learn how a hybrid mobile application works
- Familiarize yourself with Cordova and see how it fits into hybrid mobile application development
- Seamlessly work with Ionic CSS components and Ionic-AngularJS components such as directives and services
- Learn how to theme Ionic apps as well as customize components using Ionic SCSS support
- Develop an app that builds a client for a secure REST API using Ionic and AngularJS
- Develop a real-time chat app using Firebase that consumes ngCordova
- Learn how to generate a device-specific installer for an Ionic app using Ionic CLI as well as Ionic cloud services

By Mike Hartington – Core Team Member, Ionic


---

Learn how a hybrid mobile application works

Familiarize yourself with Cordova and see how it fits into hybrid mobile application development

Seamlessly work with Ionic CSS components and Ionic-AngularJS components such as directives and services

Learn how to theme Ionic apps as well as customize components using Ionic SCSS support

Develop an app that builds a client for a secure REST API using Ionic and AngularJS

Develop a real-time chat app using Firebase that consumes ngCordova

Learn how to generate a device-specific installer for an Ionic app using Ionic CLI as well as Ionic cloud services
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 2 'Welcome to Ionic'
- A synopsis of the book’s content
- More information on Learning Ionic
About the Author

**Arvind Ravulavaru** is a full-stack consultant with over 6 years of experience in software development. For the last 2 years, he has been working extensively on JavaScript, both on the server side and the client side. Prior to that, Arvind worked on big data analytics, cloud provisioning, and orchestration. He has good exposure to various databases and has also developed and architected an application that was built using Java and ASP.NET.

A year and a half ago, Arvind started a blog titled The Jackal of JavaScript ([http://thejackalofjavascript.com](http://thejackalofjavascript.com)), where he writes about developing full-stack applications using JavaScript alone as the server-side and client-side programming language. He has written articles on various topics, such as DNA analysis, sentiment analysis with JavaScript, programming Raspberry Pi with JavaScript, and building a node-webkit and a WebRTC-based video chat client, to name a few.

Apart from this, Arvind provides training, empowering companies with the latest and greatest technology available in the market. He also conducts start up workshops, where he teaches rapid prototyping with today's amazing tool stack. Arvind also provides information on how to take ideas to the market in a very short span of time.

He always looks for ways to contribute to the open source community, making this world a better place for developers. As a consultant, Arvind always tries to power amazing business ideas with awesome technology solutions (language-agnostic), moving the human race up the evolution chain.

Arvind recently started his own company in Hyderabad, where he is building products in the Internet of Things space. The aim of the company is to build IoT products for the world at an affordable price so that everyone can enjoy technology that powers lives.

You can reach Arvind on his blog at [http://thejackalofjavascript.com](http://thejackalofjavascript.com).

Arvind has also reviewed *Data-oriented Development with AngularJS, Manoj Waikar, Packt Publishing*. 
Preface

Build mobile hybrid applications with ease using Ionic. Be it simple apps that integrate REST API endpoints or complicated apps that involve native features, Ionic provides a simple API to work with them.

With a basic knowledge of HTML, CSS, and a decent knowledge of AngularJS, one can easily convert a million-dollar idea into an app with a few lines of code.

In this book, we will explore how you can do this.

What this book covers

Chapter 1, Ionic – Powered by AngularJS, introduces you to the power of AngularJS. This is why it is considered a good framework for a hybrid framework, such as Ionic.

Chapter 2, Welcome to Ionic, talks about the Mobile Hybrid framework: Cordova. It shows how Ionic fits into the bigger picture. This chapter also shows you how to install the required software for the development of the Ionic framework.

Chapter 3, Ionic CSS Components and Navigation, tells you how to use Ionic as a CSS framework for your mobile web applications. This chapter also talks about how to integrate Ionic CSS only components with AngularJS, apart from routing in Ionic apps.

Chapter 4, Ionic and SCSS, explores how to theme Ionic apps with the help of built-in SCSS support.

Chapter 5, Ionic Directives and Services, provides information on how to use Ionic's built-in directives and services, which helps to quickly bring your application to life.
Preface

Chapter 6, Building a Bookstore App, deals with how you can leverage the knowledge you have gained so far in building an Ionic client for a secured REST API. This chapter walks you through the itsy-bitsy pieces that are needed to develop Ionic apps that will simply consume REST API endpoints.

Chapter 7, Cordova and ngCordova, talks about how you can easily integrate device features, such as camera and Bluetooth, with the help of Cordova and ngCordova in your Ionic apps.

Chapter 8, Building a Messaging App, shows you how you can use all the knowledge that you have gained so far to build a messaging app, where users can register, log in, and chat with each other and share pictures and locations.

Chapter 9, Releasing the Ionic App, focuses on how to generate installers for the apps that you have built with Cordova and Ionic using the Ionic CLI and the PhoneGap build.

Appendix, Additional Topics and Tips, discusses how to efficiently use the Ionic CLI and Ionic cloud services to build, deploy, and manage your Ionic applications.

What you need for this book
To start building Ionic apps, you need to have a basic knowledge of HTML, CSS, JavaScript, and AngularJS. A good knowledge of mobile application development, device native features, and Cordova is preferred.

You will need Node.js, Cordova CLI, and Ionic CLI installed to work with Ionic. If you are planning to add theming support and other third-party libraries, you will need Git and Bower. If you want to work with device features, such as camera or Bluetooth, you need to have their mobile OS set up on your machine.

Who this book is for
This book is intended for those who want to learn how to build hybrid mobile applications using Ionic. It is also ideal for people who want to work with theming Ionic apps, integrating with the REST API, and learning more about device features, such as camera or Bluetooth, using ngCordova.

Prior knowledge of AngularJS is essential to complete this book successfully.
Welcome to Ionic

In the previous chapter, we saw a couple of key AngularJS features, namely the directives and services. In this chapter, we will understand the big picture of mobile hybrid apps, set up the required software to develop Ionic apps, and finally scaffold a few apps.

The topics covered in this chapter are as follows:

- Mobile Hybrid Architecture
- What is Apache Cordova?
- What is Ionic?
- Setting up the tools needed to develop and run Ionic apps
- Working with Ionic templates
- The Yeoman Ionic Generator

Mobile Hybrid Architecture

Before we start working with Ionic, we need to understand the bigger picture of the Mobile Hybrid platform.

The concept is pretty simple. Almost every mobile operating system (also called Platform when working with Cordova) has an API to develop apps. This API consists of a component named Web View. A Web View is typically a browser that runs inside the scope of a mobile application. This browser runs the HTML, CSS, and JS codes. This means that you can build a web page using the preceding technologies and then execute it inside your app.
You can use the same knowledge of web development to build native-hybrid mobile apps (here, native refers to installing the platform-specific format file on the device after it has been packaged along with the assets), for instance:

- Android uses Android Application Package (.apk)
- iOS uses iPhone Application Archive (.ipa)
- Windows Phone uses Application Package (.xap)

The package/installer consists of a piece of native code that initializes the web page and a bunch of assets needed to show the web page content.

This setup of showing a web page inside the mobile app container that consists of your application business logic is called as a Hybrid App.

**What is Apache Cordova?**

In simple terms, Cordova is the piece of software that stitches the web application and the native application together. The Apache Cordova website states that:

"Apache Cordova is a platform for building native mobile applications using HTML, CSS and JavaScript."

Apache Cordova does not just stitch the web app with the native app, but it also provides a set of APIs written in JavaScript to interact with the native features of the device. Yes, you can use JavaScript to access your camera, take a picture, and send it in an e-mail. Sounds exciting, right?
To get a better understanding of what is happening, let's take a look at the following image:

As you can see, we have a web view where the HTML/CSS/JS code gets executed. This code can be a simple standalone piece of user interface; at best you are making an AJAX request to get some data from a remote server. Or, this code can do much more, like talking to the Bluetooth of the device and getting the list of devices in the vicinity.

For this chapter, you can also access the code, raise issues, and chat with the author at GitHub (https://github.com/learning-ionic/Chapter-2).
Welcome to Ionic

In the latter case, Cordova has a bunch of APIs that interface with the web view using JavaScript and then talk to the device in its native language (for example, Java for Android), thus providing a bridge between them. For instance, if you would like to know more from the JavaScript about the device in which your app is running, all you need to do is write the following code inside the JS file:

```javascript
var platform = device.platform;
```

After installing the device plugin, you can also access the UUID, model, OS version, and the Cordova version of the device from inside the web view using JavaScript as follows:

```javascript
var uuid = device.uuid;
var model = device.model;
var version = device.version;
var Cordova = device.Cordova;
```

We will deal more with Cordova plugins in Chapter 7, Cordova and ngCordova.

The preceding explanation was to give you an idea of how Mobile Hybrid apps are structured and how you can use device features from the web view using JavaScript.

Cordova does not convert the HTML, CSS, and JS code to an OS-specific binary code. All it does is wrap the HTML, CSS, and JS code and execute it inside a web view.

So, you must have guessed by now that Ionic is the framework with which we build the HTML/CSS/JS code that runs in the web view and talks with Cordova to access device specific APIs.

What is Ionic?

Ionic is a beautiful, open source, front-end SDK for developing hybrid mobile apps with HTML5. Ionic provides mobile-optimized HTML, CSS, and JS components, as well as gestures and tools for building highly interactive apps.

Ionic is performance efficient with its minimal DOM manipulation and hardware-accelerated transitions as compared to other frameworks in this league. Ionic uses AngularJS as its JavaScript framework.

With the power of AngularJS inside a framework like Ionic, the possibilities are unlimited (you can use any AngularJS component inside Ionic as long as it makes sense in a mobile app). Ionic has a very good integration with Cordova’s device API. This means that you can access device APIs using a library like ngCordova and integrate it with the beautiful user interface components of Ionic.
Ionic has its own Command Line Interface (CLI) to scaffold, develop, and deploy Ionic apps. Before we start working with the Ionic CLI, we need to set up a few pieces of software.

**Software setup**

Now we are going to set up all the required software needed to develop and run an Ionic app smoothly.

**Install Node.js**

Since Ionic uses Node.js for its CLI as well as for the build tasks, we will first install the same as follows:

1. Navigate to https://nodejs.org/.
2. Click on the Install button on the homepage and an installer for your OS will automatically be downloaded. Or you can navigate to https://nodejs.org/download/ and download a specific copy.
3. Install Node.js by executing the downloaded installer.

To verify that Node.js has been successfully installed, open a new Terminal (*nix systems) or Prompt (Windows systems) and run the following command:

```bash
node -v
```

Now you should see the version of Node.js. Now execute the following command:

```bash
npm -v
```

You should see the npm version:

```
$ node -v
v0.12.2
$ npm -v
2.10.0
```

You need Node.js only during the development. The version specified in the preceding image is only for illustration. You may have the same version or the latest version of the software. This applies to all the images that show the software version in this chapter.
Welcome to Ionic

Install Git
Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. In our case, we will be using a package manager named Bower, which uses Git to download the required libraries. Also, the Ionic CLI uses Git to download the project templates.

To install Git, navigate to http://git-scm.com/downloads and download the installer for your platform. Once you have successfully installed it, you can navigate to your terminal/prompt and run the following command:

```
$ git --version
```

You should see the following output:

```
+ ~ git --version
git version 2.3.2 (Apple Git-55)
```

Install Bower
We are going to use Bower (http://bower.io/) to manage our application library dependencies. Bower is a package manager similar to npm but a linear/flat version of it. This kind of package manager is more suitable for downloading the assets needed for web development.

Bower is built on top of Node.js. To install Bower globally from the terminal/prompt you need to run the following command:

```
npm install bower -g
```

We are installing a node module named Bower globally. Hence, the `-g` flag. On *nix systems, you may need to use `sudo` to run this command:

```
sudo npm install bower -g
```

If you need `sudo` for running the preceding command, please recheck your npm installation. Refer to http://competa.com/blog/2014/12/how-to-run-npm-without-sudo/ for more information on `sudo` less npm global installs.
Once Bower is successfully installed, you can verify the same by running the following command:

```
bowerr -v
```


to install gulp globally run the following command:

```
npm install gulp -g
```

For *nix systems, run this:

```
sudo npm install gulp -g
```

Once Gulp is successfully installed, you can verify the same by running this command:

```
gulp -v
```

Install Sublime Text

This is a totally optional installation. Everyone has their own preferred text editor. After running around many text editors, I fell in love with Sublime Text, purely for its simplicity and the number of Plug and Play packages.
Welcome to Ionic

If you would like to give this editor a try, you can navigate to http://www.sublimetext.com/3 to download Sublime Text 3.

Install Cordova and Ionic CLI

Finally, to complete the Ionic setup, we will install the Ionic CLI. Ionic CLI is a wrapper around the Cordova CLI with some additional features.

All the code examples in this book use Cordova version 5.0.0, Ionic CLI version 1.5.0, and Ionic version 1.0.0 (uranium-unicorn).

To install the Ionic CLI run the following command:

    npm install cordova@5.0.0 ionic@1.5.0 -g

To verify the install run the following command:

    cordova -v

You can also run this:

    ionic -v

```
    ➜  ~ cordova -v
    5.0.0
    ➜  ~ ionic -v
    1.5.0
```

To get a feel of what Ionic CLI is packed with, run:

    ionic
You should see a list of tasks as seen in the following screenshot:

```
+ ~ ionic

Usage: ionic task args

Available tasks: (use --help or -h for more info)

start ........... Starts a new Ionic project in the specified PATH
serve ............ Start a local development server for app dev/testing
platform ......... Add platform target for building an Ionic app
run ............... Run an Ionic project on a connected device
emulate .......... Emulate an Ionic project on a simulator or emulator
build ............. Locally build an Ionic project for a given platform
plugin ............ Add a Cordova plugin
resources ....... Automatically create icon and splash screen resources (beta)
                Put your images in the ./resources directory, named splash or icon.
                Accepted file types are .png, .ai, and .psd.
                Icons should be 192x192 px without rounded corners.
                Splashscreens should be 2208x2208 px, with the image centered in the middle.
```

There are a few more tasks apart from the ones seen in the preceding screenshot.

You can read through the tasks and explanations to get an idea about what they do. Also, note that some of the tasks are still in beta as of today.

With this, we have completed the installation of all the software needed to develop apps with Ionic.

**The platform guide**

By the end of this book, we will be building apps that are ready to be deployed on to the device. Since Cordova takes HTML, CSS, and JS code as input and generates a platform-specific installer, you need to have the build environments available on your machine.
Welcome to Ionic


iOS users can follow the instructions from the iOS Platform Guide at http://cordova.apache.org/docs/en/edge/guide_platforms_ios_index.md.html#iOS%20Platform%20Guide to set up SDK on your local machine.

You would need an OSX environment to develop iOS apps.

As of today, Ionic supports only Android 4.0+ (although, it works on 2.3 as well) and iOS 6+ mobile platforms. But Cordova supports a few more.

You can check out the other supported platforms here: http://cordova.apache.org/docs/en/edge/guide_platforms_index.md.html#Platform%20Guides.

Hello Ionic

Now that we are done with the software setup, we will scaffold a few Ionic apps.

Ionic has three main/go-to templates using which we can quickly start developing apps:

- **Blank**: This is a blank Ionic project with one page
- **Tabs**: This is a sample app that is built using Ionic tabs
- **Side menu**: This is a sample app that is built to consume side menu driven navigation

To understand the basics of scaffolding, we will start with the blank template.

To keep our learning process clean, we will create a folder structure to work with Ionic projects. Create a folder named ionicApps, and then create a folder inside it called chapter2.

Next, open a new terminal/prompt and change the directory (cd) to the ionicApps folder and from there to chapter2 folder. Now run the following command:

`ionic start -a "Example 1" -i app.example.one example1 blank`
In the preceding command:

- `-a "Example 1"`: This is the human-readable name of the app
- `-i app.example.one`: This is the app ID/reverse domain name
- `example1`: This is the name of the folder
- `blank`: This is the name of the template

Refer to the Appendix, Additional Topics and Tips, to know more about the Ionic start task.

The Ionic CLI is very verbose when it comes to performing tasks. As you can see from the terminal/prompt, while the project is getting created a lot of information is printed.

To start off, a new blank project is downloaded and saved to the `example1` folder. Next, the `ionic-app-base` is downloaded from the `ionic-app-base` GitHub repo, https://github.com/driftyco/ionic-app-base. Post that, `ionic-starter-template` is downloaded from the `ionic-starter-template` GitHub repo at https://github.com/driftyco/ionic-starter-blank.

After that, the config file is updated with the app name and ID. Next, a script runs and five Cordova plugins are downloaded:

- `org.apache.cordova.device` (https://github.com/apache/cordova-plugin-device): This is used to get device information, as we have seen earlier in this chapter
- `org.apache.cordova.console` (https://github.com/apache/cordova-plugin-console): This plugin is meant to ensure that `console.log()` is as useful as it can be
- `cordova-plugin-whitelist` (https://github.com/apache/cordova-plugin-whitelist): This plugin implements a whitelist policy for navigating the application web view on Cordova 4.0
- `cordova-plugin-splashscreen` (https://github.com/apache/cordova-plugin-splashscreen): This plugin shows and hides a splash screen during the application launch
- `com.ionic.keyboard` (https://github.com/driftyco/ionic-plugins-keyboard): This is the keyboard plugin providing functions to make interaction with the keyboard easier, and fires events to indicate whether the keyboard will hide/show
Welcome to Ionic

All this information is then added to the package.json file and a new ionic.project file is created.

Once the project has been successfully created, you will see a bunch of instructions on how to proceed further. Your output should look something like the following screenshot:

Your Ionic project is ready to go! Some quick tips:
• cd into your project: $ cd example1
• Setup this project to use Sass: ionic setup sass
• Develop in the browser with live reload: ionic serve
• Add a platform (ios or Android): ionic platform add ios [android]
  Note: iOS development requires OS X currently
  See the Android Platform Guide for full Android installation instructions:
  https://cordova.apache.org/docs/en/edge/guide_platforms_android_index.md.html
• Build your app: ionic build <PLATFORM>
• Simulate your app: ionic emulate <PLATFORM>
• Run your app on a device: ionic run <PLATFORM>
• Package an app using Ionic package service: ionic package <MODE> <PLATFORM>
For more help use ionic --help or ionic docs
Visit the Ionic docs: http://ionicframework.com/docs

New! Add push notifications to your Ionic app with Ionic Push (alpha):
https://apps.ionic.io/signup
+ New Ionic Updates for June 2015
+ The View App just landed. Preview your apps on any device
  http://view.ionic.io
+ Invite anyone to preview and test your app
  ionic share EMAIL
+ Generate splash screens and icons with ionic resource
  http://ionicframework.com/blog/automating-icons-and-splash-screens/

To proceed further, we will use the cd command to go to the example1 folder. We will not follow the instructions provided in the terminal/prompt, as we still understand the project setup. Once we have a fair idea on the various components in Ionic, we can start using the commands from the terminal/prompt output after we scaffold a new Ionic App.
Once we have changed directory to `example1` folder, we will serve the app by giving the following command:

```
ionic serve
```

This will start a new dev server on port `8100`, and then launch the app in your default browser. I highly recommend setting Google Chrome or Mozilla Firefox as your default browser while working with Ionic.

When the browser launches, you should see the blank template.

If you run this:

```
ionic serve
```

You will see an error as shown here:

```
* example1 ionic serve
The port 35729 was taken on the host localhost - using port 35730 instead
Running live reload server: http://localhost:35730
Watching : ['www/**/*', 'www/lib/**/*']
Running dev server: http://localhost:8100
Ionic server commands, enter:
  restart or r to restart the client app from the root
  goto or g and a url to have the app navigate to the given url
  consolelogs or c to enable/disable console log output
  serverlogs or s to enable/disable server log output
  quit or q to shutdown the server and exit

ionic $ An uncaught exception occurred and has been reported to Ionic

listen EADDRINUSE (CLI v1.5.0)

Your system information:

Cordova CLI: 5.0.0
Gulp version: CLI version 3.8.11
Gulp local:
  Ionic Version: 1.0.0
  Ionic CLI Version: 1.5.0
  Ionic App Lib Version: 0.1.0
  ios-deploy version: 1.7.0
  ios-sim version: 3.1.1
OS: Mac OS X Yosemite
Node Version: v0.12.2
Xcode version: Xcode 6.3.2 Build version 6D2105
```

This means that some other application on your machine is running on the port `8100`. To fix this, you can use any other port like `8200` while running Ionic serve:

```
ionic serve -p 8200
```
Once the application is successfully launched and we have seen the output in the browser, we will navigate back to the terminal/prompt and we should see something like the following screenshot:

```
+ example1 ionic serve
Running live reload server: http://localhost:35729
Watching : [ 'www/**/*', '!www/lib/**/*' ]
Running dev server: http://localhost:8100
Ionic server commands, enter:
    restart or r to restart the client app from the root
    goto or g and a url to have the app navigate to the given url
    consolelogs or c to enable/disable console log output
    serverlogs or s to enable/disable server log output
    quit or q to shutdown the server and exit

ionic $ 
```

As mentioned earlier, Ionic CLI tasks are pretty verbose. They will not leave you hanging. You can see that while the Ionic server command is running, you can type R and hit Enter and the application restarts. Similarly, you can press C to enable or disable printing browser JavaScript logs to the terminal/prompt.

Once you are done running the application, press Q and hit Enter to stop the server. You can do the same by pressing Ctrl + C from the keyboard.

**The browser developer tools setup**

Before we proceed further, I would recommend setting up the developer tools in your browser in the following format.

**Google Chrome**

Once the Ionic app is launched, open the developer tools by pressing Cmd + Opt + 1 on Mac and Ctrl + Shift + 1 on Windows/Linux. Then click on the last but one icon in the top row, next to the close button as seen in the following screenshot:
This will dock developer tools to the side of the current page. Drag the demarcation line between the browser and the developer tools till you see the view becoming similar to a mobile.

If you click on the **Elements** tab in the developer tools, you can easily inspect the page and see the output in one go, as shown in the following screenshot:

![Developer Tools Screenshot]

This view is very helpful for fixing errors and debugging issues.
Mozilla Firefox

If you are a Mozilla Firefox fan, you can achieve the same result as above as well. Once the Ionic app is launched, open developer tools (not Firebug, Firefox’s native development tools) by pressing `Cmd + Opt + I` on Mac and `Ctrl + Shift + I` on Windows/Linux. Then click on the Dock to side of browser window icon, as shown in the following screenshot:

![Mozilla Firefox Developer Tools](image)

Now, you can drag the demarcation line to achieve the same result as we have seen in Chrome:
The Ionic project structure

So far, we have scaffolded a blank Ionic app and launched it in a browser. Now, we will walk through the scaffolded project structure.

To quickly remind you, we know that the Ionic sits inside the Cordova application. So before we go to the Ionic code, we will talk about the Cordova wrapper.

If you open the `chapter2` `example1` folder in your text editor, you should see the following folder structure at the root of the project:

```
├── bower.json
├── config.xml
├── gulpfile.js
├── hooks
├── ionic.project
├── package.json
├── plugins
├── scss
└── www
```

Here is a quick explanation of each of the items:

- `bower.json`: This consists of the dependencies loaded via Bower. In future, we will install other Bower packages to be used along with our app. All the information regarding the same will be available here.
- `config.xml`: This file consists of all the meta information needed by Cordova while converting your Ionic app to a platform-specific installer. If you open `config.xml`, you will see a bunch of XML tags that describes your project. We will take a look at this file in detail again.
- `gulpfile.js`: This file consists of the build tasks that we would be using while developing the Ionic app.
- `ionic.project`: This file consists of the information regarding the Ionic app.
Welcome to Ionic

- **hooks**: This folder consists of scripts that get executed when a particular Cordova task is performed. A Cordova task can be any of the following: `after_platform_add` (after a new platform is added), `after_plugin_add` (after a new plugin is added), `before_emulate` (before emulation begins), `after_run` (before the app run), and so on. Each task is placed inside a folder named after the Cordova task. If you open the `hooks` folder, you will see an `after_prepare` folder and a `README.md` file. Inside the `after_prepare` folder, you will find a script file named `010_add_platform_class.js`. This will get executed after the prepare task of Cordova is executed. All that this task does is add a class to the `<body>` tag, which is the same name as the platform on which the app is running. This would help us in styling the app better, based on the platform. You can find a list of tasks that you can hook into from the `README.md` file present inside the `hooks` folder.

- **plugins**: This folder consists of all the plugins added to the project. We will be adding a few more plugins later on, and you can see them reflected here.

- **scss**: This folder consists of the base `scss` file which we will be overwriting to customize the styles of the Ionic components. More on this in Chapter 4, *Ionic and SCSS*.

- **www**: This folder consists of the Ionic code. Anything you write inside this folder is intended to land inside the web view. This is where we will be spending most of our time.

**The config.xml file**

The `config.xml` file is a platform agnostic configuration file. As mentioned earlier, this file consists of all the information needed by Cordova to convert the code in the `www` folder to the platform-specific installer.

Setting up of the `config.xml` file is based on the W3C’s Packaged Web Apps (Widgets) specification ([http://www.w3.org/TR/widgets/](http://www.w3.org/TR/widgets/)) and extended to specify core Cordova API features, plugins, and platform-specific settings. There are two types of configurations that you can add to this file. One is global, that is, common to all devices, and the other is specific to the platform.

If you open `config.xml` in your text editor, the first tag you encounter is the XML root tag. Next, you can see the widget tag:

```xml
<widget id="app.example.one" version="0.0.1" xmlns="http://www.w3.org/ns/widgets" xmlns:cdv="http://cordova.apache.org/ns/1.0"/>
```
The id specified above is the reverse domain name of your app, which we provided while scaffolding. Other specifications are defined inside the widget tag as its children. The children tags include the app name (that gets displayed below the app icon when installed on the device), app description, and author details.

It also consists of the configuration that needs to be adhered to while converting code in the www folder to a native installer.

The content tag defines the starting page of the application. The access tag defines the URLs that are allowed to load in the app. By default, it loads all the URLs. The preference tag sets the various options as name value pairs. For instance, DisallowOverscroll describes if there should be any visual feedback when the user scrolls past the beginning or end of the document.

You can read more about platform-specific configurations at the following links:

- **Android:** [http://docs.phonegap.com/en/4.0.0/guide_platforms_android_config.md.html#Android%20Configuration](http://docs.phonegap.com/en/4.0.0/guide_platforms_android_config.md.html#Android%20Configuration)
- **iOS:** [http://docs.phonegap.com/en/4.0.0/guide_platforms_ios_config.md.html#iOS%20Configuration](http://docs.phonegap.com/en/4.0.0/guide_platforms_ios_config.md.html#iOS%20Configuration)

The importance for the platform-specific configurations and global configuration is same. You can read more about global configuration at [http://docs.phonegap.com/en/4.0.0/config_ref_index.md.html#The%20config.xml%20File](http://docs.phonegap.com/en/4.0.0/config_ref_index.md.html#The%20config.xml%20File)

### The www folder

As mentioned earlier, this folder consists of our Ionic app, the HTML, CSS, and JS codes. If you open the www folder, you will find the following file structure:

```
.
├── css
│   └── style.css
├── img
│   └── ionic.png
├── index.html
├── js
│   └── app.js
└── lib
    └── ionic
```
Welcome to Ionic

- css
- fonts
- js
- scss
- version.json

Let's look at each of these in detail:

- **index.html**: This is the application startup file. If you remember, in `config.xml`, we pointed the `src` on the content tag to this file. Since we are using AngularJS as our JavaScript framework, this file would ideally act as the base/first page for the **Single Page Application (SPA)**. If you open `index.html`, you can see that the body tag has the ng-app attribute which points to the starter module defined inside the `js/app.js` file.

- **css**: This folder consists of styles that are specific to our app.

- **img**: This folder consists of images that are specific to our app.

- **js**: This folder consists of the JavaScript code specific to our app. This is where we write the AngularJS code. If you open the `app.js` file present inside this folder, you can see that an AngularJS module is set up, passing in Ionic as a dependency.

- **lib**: This folder is where all the packages would be placed when we run `bower install`. When we scaffold the app, this folder comes along with it, loaded with Ionic files. If you want to download the assets again along with its dependencies, you can use the `cd` command to go to the `example1` folder from your terminal/prompt and run the following command:

  `bower install`

You will see that four more folders get downloaded. These are a part of the dependencies listed in the `ionic-bower` package, present in `bower.json` file, located at the root of the project.

Ideally, we would not be using these libraries explicitly in our app. Rather, we would use the Ionic bundle which is built on top of these libraries.

This completes our tour of the blank template. Before we scaffold the next template, let us take a quick peek at the `www/js/app.js` file.
As you can see, we are creating a new AngularJS module with the name starter, and then we are injecting ionic as a dependency.

The $ionicPlatform service is injected as a dependency to the run method. This service is used to detect the current platform, as well as handle the hardware buttons on (Android) devices. In the current context, we are using the $ionicPlatform.ready method to be notified when the device is ready for us to perform operations on it.

It is a good practice, or rather a necessity in some cases, to include all your code inside the $ionicPlatform.ready method. This way, your code executes only when the entire app is initialized.

So far you must have worked on AngularJS code related to the Web. But when you are dealing with Ionic, you would be working with scripts related to device features as well, inside the AngularJS code. Ionic provides us services to make these things happen in a more organized fashion. Hence, we went through the concept of custom services in Chapter 1, Ionic – Powered by AngularJS, and we will be going through Ionic Services in depth in Chapter 5, Ionic Directives and Services.

**Scaffolding the tabs template**

To get a good feel of the Ionic CLI and the project structure, we will scaffold the other two starter templates as well. First we will scaffold the tabs template.

Using the cd command go back to the chapter2 folder and run the following command:

    ionic start -a "Example 2" -i app.example.two example2 tabs

As you can see, the tabs project is scaffolded inside the example2 folder. Using the cd command go to the example2 folder and execute the following command:

    ionic serve
Welcome to Ionic

You should see the tabbed interface app built using Ionic as seen in the following screenshot:

![Tabbed Interface App](image)

The tabs are located at the bottom of the page. We will talk more about customizations in Chapter 3, Ionic CSS Components and Navigation and Chapter 5, Ionic Directives and Services.

If you go back to the example2 folder and analyze the project structure, everything would be the same except for the contents of the www folder.

This time, you will see a new folder named templates. This folder will consist of partial pages associated with each AngularJS route. And if you open the js folder, you will find two new files:

- **controller.js**: This consists of the AngularJS controller code
- **services.js**: This consists of the AngularJS service code

Now you can get a good idea of how Ionic is integrated with AngularJS, and how all the components go hand-in-hand. When we deal with a few more pieces of Ionic, this structure will make a lot of sense.
Scaffolding the side menu template

Now, we will scaffold the final template. Using the `cd` command go back to the `chapter2` folder and run the following command:

```
ionic start -a "Example 3" -i app.example.three example3 sidemenu
```

To execute the scaffolded project, using the `cd` command go to the `example3` folder and give the following command:

```
ionic serve
```

And the output should be similar to the following screenshot:

You can analyze the project structure yourself and see the difference.

You can run `Ionic start -l` or `ionic templates` to view the list of available templates. You can also use the `ionic start` task with the template name from the list to scaffold the app.
Welcome to Ionic

**generator-ionic**

I feel that the Ionic CLI is pretty good, but it does not have a workflow associated with it. When I say workflow, I mean a demarcation between the development code and production code. In the project scaffolded by the Ionic CLI, the `www` folder hosts both the development code and production code. This will become an issue very fast when your app starts growing.

This is where `generator-ionic` proves to be invaluable. The `generator-ionic` is a Yeoman generator that is used to scaffold Ionic projects. If you did not already know this, Yeoman is a scaffolding tool that uses Grunt, Bower, and Yo to scaffold apps. Of late, they've started supporting gulp as well.

### Why Yeoman?

Unlike the IDEs for other languages, JavaScript or web development does not have a unified development environment, where a user would navigate to **File** | **New** | AngularJS project or **File** | **New** | HTML5 project. This is where Yeoman fits in.

Ionic has its own CLI to scaffold apps. But for other frameworks, which do not have a generator, Yeoman provides base generators.


There are other generators available for Ionic, but I prefer `generator-ionic` ([https://github.com/diegonetto/generator-ionic](https://github.com/diegonetto/generator-ionic)) for its workflow and features.

### Installing generator-ionic

Before we install `generator-ionic`, we need to install `yo`, `grunt`, and `grunt-cli` globally. To do this, run the following command:

```bash
npm install yo grunt grunt-cli -g
```

Grunt is another build tool similar to Gulp. The main difference between Grunt and Gulp is that, Grunt is a configuration-over-code build tool, whereas gulp is a code-over-configuration build tool.
You can read more about Grunt at http://gruntjs.com/.
You can find the presentation from my talk on Gulp versus Grunt at http://arvindr21.github.io/building-n-Scaffolding for some more insights into them.

Next, we will install generator-ionic globally:

```shell
npm install generator-ionic -g
```

Packages which are installed with a -g flag need to be installed only once. You need not install them again and again for each usage.

Now, we can scaffold a new Ionic project using generator-ionic. Using the cd command go back to the chapter2 folder and create a new folder inside it named example4. Run the following command:

```shell
yo ionic example
```

Unlike the Ionic CLI, you need to answer a few questions on how you would like to scaffold your app. You can answer the questions as shown below:

? Would you like to use Sass with Compass (requires Ruby)?
  N

? Which Cordova plugins would you like to include?
  org.apache.cordova.device
  org.apache.cordova.console
  com.ionic.keyboard

? Which starter template would you like to use?
  Tabs

Yeoman will go ahead and download all the stuff needed for the project to run. Once the Yeoman scaffolding is complete, you can navigate to the example4 folder. As you can see, there are a lot more files and folders present in this folder.

You can read more about the complete project structure at https://github.com/diegonetto/generator-ionic#project-structure.
Welcome to Ionic

The key differences between the Ionic CLI scaffolded app and the generator-ionic scaffolded app are as follows:

- **app**: unlike the Ionic CLI scaffolded app, we will carry out the development not inside the www folder but inside the app folder. This was the code demarcation that I was referring to. We develop inside the app folder and run a build script that takes care of cleaning up the files and putting them inside the www folder, ready for production use.

- **hooks**: If you open the hooks folder, you can find four more scripts.

- **Gruntfile.js**: Unlike the Ionic CLI, generator-ionic uses Grunt to manage tasks. If you feel that there are too many things to learn, I recommend you follow the Ionic CLI scaffolded app over generator-ionic and Gulp over Grunt.

If you are using generator-ionic to scaffold your app, do not work inside the www folder. When you run the build command, the contents of this folder will be deleted and recreated from the app folder.

You can check out the workflow commands that you can use to run the app from [https://github.com/diegonetto/generator-ionic#workflow-commands](https://github.com/diegonetto/generator-ionic#workflow-commands).

All the Ionic CLI methods are wrapped with the grunt command. So, for instance, when you want to execute Ionic serve, you will run grunt serve when using generator-ionic.

So, let's serve the scaffolded app by running the following command:

```
grunt serve
```

You should see the same output that you have seen for the tabs app when we scaffolded it with the Ionic CLI.

Three more reasons that made me choose generator-ionic over Ionic CLI is the out-of-the-box support for:

- **Code hinting** ([https://github.com/diegonetto/generator-ionic#grunt-jshint](https://github.com/diegonetto/generator-ionic#grunt-jshint))

- **Testing with Karma** (testing framework) and code coverage using Istanbul ([https://github.com/diegonetto/generator-ionic#grunt-karma](https://github.com/diegonetto/generator-ionic#grunt-karma))

- **Ripple emulator** ([https://github.com/diegonetto/generator-ionic#grunt-ripple](https://github.com/diegonetto/generator-ionic#grunt-ripple))
Chapter 2

The main reason I am demonstrating generator-ionic is to introduce you to the workflow so that you can adapt to it when your app becomes bigger. Again, this is a personal preference and you may like to work with the Ionic CLI itself.

You can play around with other Ionic generators as well and see which you are comfortable with.

Summary

In this chapter, you gained some knowledge on mobile hybrid architecture. You also learned how a Hybrid app works. We saw how Cordova stitches the HTML, CSS, and JS code to be executed inside the web view of the app. Then, we installed the required software to develop Ionic apps locally. We scaffolded a blank template using the Ionic CLI and analyzed the project structure. Later on, we scaffolded the other two templates and observed the difference. We also installed generator-ionic and scaffolded a sample app, and observed the difference between a generator-ionic and Ionic CLI scaffolded app.

You can also refer to Ionic slides at http://ionicframework.com/present-ionic/slides for some more information.

In the next chapter, we will understand Ionic CSS components and the router. This will help us in building interesting user interfaces and multipage applications using the Ionic API.
Where to buy this book
You can buy Learning Ionic from the Packt Publishing website.
Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.
Click here for ordering and shipping details.